Deep learning - 1. Assignment - Basic Neural Network

Nejc Ločičnik (63180183)

1 Introduction

This assignment is about basic neural networks, how they functions, their basic properties, how to improve results and what kinda of hyper parameters are generally available for fine-tuning.

2 Basic implementation

The implementation is based on the pre-available template, which was adapted a bit. The major things that were changed is initialization of biases, which were initially set to 0. They are initialized with a normal distribution with a small standard deviation just to reduce the symmetry at the start of training. The second thing is that the samples are shuffled before each epoch, so we don't allow the network to learn from the fixed sequence of images.

2.1 Architecture

After a bit of testing a network structure with 3 hidden layers, each with 100 neurons seemed to work best ([in, 100, 100, 100, out]). I tried increasing/decreasing both the amount of hidden layers and neurons and I generally got worse results.

2.2 Activation function

The required activation function in the hidden layers wasn't specified so I tried a few things. I started with sigmoid, which I then change to a numerically stable version of sigmoid function and finally ReLu, which proved to be the best.

The results for the base implementation are presented in table **??**. The used optimizer is always SGD for these experiments. The first columns ("Last h. l.) is regarding the last/third hidden layer, the first two always have 100 neurons. Experiments that reduced performance aren't written.

Last h. l.	Epochs	Activation	Learning rate	Accuracy
/	10	sigmoid	0.05	0.4218
/	10	sigmoid	0.075	0.4302
50	10	sigmoid	0.25	0.4489
100	10	sigmoid	0.25	0.4607
100	10	relu	0.025	0.4852
100	20	relu	0.025	0.5032
100	50	relu	0.02	0.519

Table 1: Results of the base implementation of the NN.

3 Regularization

Regularization didn't seem to improve result. I implemented the basic L2 regularization. I tried using a fixed and adaptive regularization parameter (e.g. lambda*log(epoch)).

Figure 1 and 2, showcase how training loss and accuracy don't simply run away from the loss and accuracy of outside (evaluation) data. We can clearly see how regularization generalises the training. I used a lambda of 0.5.

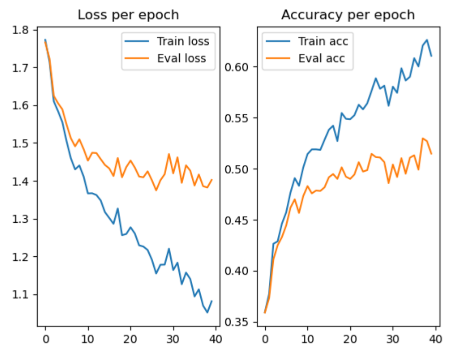

Figure 1: Loss and accuracy without regularization.

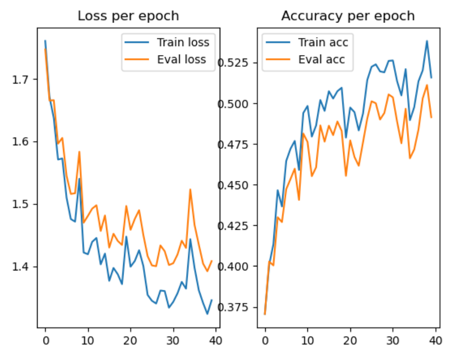

Figure 2: Loss and accuracy with adaptive regularization.

4 ADAM optimizer

I didn't make many experiments here, I followed the implementation in the original paper. One thing to note is that the learning rate needs to be around 1 percent of what it is for SGD (best I found was 0.0002). The performance generally didn't improve much, but the speed of convergence, specially in the first 3 epochs is a lot faster.

5 Learning rate schedule

Here I experimented with a few things. Other than the exponential learning schedule, I implemented a few other types (linear, cosine and inverse square), mostly because I didn't like how small the learning rate gets in later epochs with exponential schedule. Personally I liked linear the best.

An observation here is that ADAM optimizer doesn't benefit much from a learning rate schedule, probably because it's already very good at maneuvering the space. SGD on the other hand benefits a lot, it allows it to speed up initial epochs so it's comparable to ADAM.

6 Conclusion

The additional improvements such as regularization, ADAM and learning rate schedule didn't result in an increase in performance, but they did improve robustness and speed of training. With them I can reliably hit 50+ percent accuracy, with lesser amount of epochs.

I think the problem is too simple and amount of data is not large enough for the improvements to really show their prowess.